An Interclass Margin Maximization Learning Algorithm for Evolving Spiking Neural Network

An Interclass Margin Maximisation Learning Algorithm for Evolving Spiking Neural Network

Shirin Dora, Sundaram Suresh and Narasimhan Sundararajan

***Abstract*—This paper presents a new learning algorithm developed for a three layered spiking neural network for pattern classification problems. The learning algorithm maximizes the interclass margin and is referred to as the Two Stage Margin Maximization Spiking Neural Network (TMM-SNN). In the first stage (structure learning stage), the learning algorithm completely evolves the hidden layer neurons in the first epoch. Further, in this stage, TMM-SNN updates the weights of the hidden neurons for multiple epochs using the newly developed normalized membrane potential learning rule such that the interclass margins (based on the response of hidden neurons) are maximized. The normalized membrane potential learning rule considers both the local information in the spike train generated by a presynaptic neuron and the existing knowledge (synaptic weights) stored in the network to update the synaptic weights. After the first stage, the number of hidden neurons and their parameters are not updated. In the second stage (output weights learning stage), TMM-SNN updates the weights of the output layer neurons for multiple epochs to maximize the interclass margins (based on the response of output neurons). Performance of TMM-SNN is evaluated using ten benchmark data sets from the UCI machine learning repository. Statistical performance comparison of TMM-SNN with other existing learning algorithms for SNNs is conducted using the nonparametric Friedman test followed by a pairwise comparison using the Fisher's least significant difference method. The results clearly indicate that TMM-SNN achieves better generalization performance in comparison to other algorithms.**

***Index Terms*—classification, multilayer network, spiking neural networks**

I. Introduction

Many studies in the neuroscience literature have presented evidence that suggest the use of spike times for representing information in the brain [1], [2]. These results have led to the development of the third generation of artificial neural networks, known as spiking neural networks [3]. A spiking neural network consists of spiking neurons that emulate the behavior of biological neurons. Studies on the computational power of spiking neurons have shown that they are more powerful than sigmoidal neurons used in earlier network architectures [4]. This has motivated researchers to develop new and efficient learning algorithms for Spiking Neural Networks (SNNs).

SpikeProp [5] is one of the earliest learning algorithms developed for SNNs. It updates the weights using the gradient of an error function which is based on the difference between the actual and desired spike times of the output neurons. Several extensions to SpikeProp have been proposed to improve its convergence [6] and also for handling multiple spikes [7], [8]. SpikeProp and its extensions are susceptible to what is referred to as the 'silent neuron problem' [9] where one faces the difficulty of computing the gradient when a spiking neuron does not generate a spike due to previous weight updates.

Another well-known approach that has been used to develop several learning algorithms for SNNs is the scheme of rank order learning [10]. Rank order learning uses the order of first spikes generated by the presynaptic neurons instead of their precise spike times to update the synaptic weights in the network. Several variants of rank order learning have been developed [11], [12], [13], [14], [15], [16], [17], [18]. In [11], rank order learning has been used to develop an evolving Spiking Neural Network (eSNN) model for processing of audio visual information. The model has two separate networks that process visual and audio information respectively. The output layer of these modality specific networks is connected to another layer of spiking neurons that responds to information from both modalities. The network has been used for the task of person authentication using face images as visual input and spoken phrases as audio input. In [12], rank order learning has been used with Spike Driven Synaptic Plasticity (SDSP) to adapt weights in an eSNN. Rank order learning is used to initialize the weights of these synapses based on the first spike and then SDSP is used to adapt the weights of the synapses based on subsequent spikes. In [13], [14], rank order learning has been extended to use the precise spike times of the presynaptic neurons along with the order of their presynaptic spikes. Other extensions to rank order learning [15], [16], [17] focus on improving the convergence of rank order learning. A detailed survey of various rank order learning based algorithms and their applications has been provided in [19]. Both rank order learning and SpikeProp schemes are non-local learning techniques. Rank order learning requires information about the timing of all presynaptic spikes to a neuron in order to determine the rank of a given spike. Similarly SpikeProp requires information about the weights of other synapses in order to update the weights of a given synapse.

Spike Timing Dependent Plasticity (STDP) is a biologically observed plasticity mechanism that relies only on the locally available information to update the synaptic weights. It uses the difference between the pre- and postsynaptic spike times to update the weights of a given synapse. STDP is an extension of Hebbian learning to spike based signals. Similar to other formulations of Hebbian learning, STDP is also inherently unstable due to its local nature [20], [21]. It has been used in combination with other learning techniques to develop learning algorithms for SNNs. One such learning algorithm is Synaptic

Shirin Dora (shirin002@e.ntu.edu.sg), Sundaram Suresh (ssundaram@ntu.edu.sg) and Narasimhan Sundararajan(ensundara@ntu.edu.sg) are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore

Weight Association Training (SWAT) [22] which uses STDP with the Bienenstock Cooper Munro [23] learning rule for updating the synaptic weights in a SNN. It modulates the height of the plasticity window in STDP using the Bienenstock Cooper Munro learning rule to ensure convergence. Remote Supervised Method (ReSuMe) [24] is another STDP based approach that combines STDP with anti-STDP to update the synaptic weights in a SNN.

For handling pattern classification problems using SNNs, most of the existing gradient and STDP based learning algorithms choose a desired spike pattern and then update the synaptic weights to minimize the difference between the actual and the desired spike pattern. This approach does not take into account the overlap between classes while updating the synaptic weights. In a classification problem, it is important to handle the overlap between classes to efficiently approximate the decision boundary. Hence, instead of learning to generate a desired spike pattern, in this paper, the weights of a SNN are updated such that the interclass margin is maximized. For this purpose, in this paper, a new normalized membrane potential learning rule has been developed. The normalized membrane potential learning rule also relies only on the locally available information to update the weights of a given synapse, but it is stable in nature. The normalized membrane potential learning rule utilizes the normalized contribution of a presynaptic neuron towards the membrane potential and the previously learnt information stored in the network to update the synaptic weights.

Based on this normalized membrane potential learning rule, in this paper, a Two stage Margin Maximization Spiking Neural Network (TMM-SNN) is presented along with its two stage learning algorithm. TMM-SNN employs a three layered SNN with an evolving hidden layer. In the first stage (*structure learning stage*), the learning algorithm evolves the hidden layer neurons and estimates the synaptic weights of the hidden neurons. At the end of this stage, the hidden neuron parameters are fixed and are not updated again. In the second stage (*output weights learning stage*), the learning algorithm estimates the weights for output layer neurons.

In the *structure learning stage*, the learning algorithm employs two different learning strategies, namely 'neuron addition strategy' and 'margin maximization strategy for structure learning'. In the first epoch, the learning algorithm adds a neuron to the network when the criterion for 'neuron addition strategy' is satisfied. The synaptic weights and threshold for the newly added neuron are initialized using the normalized contribution of the presynaptic neurons towards the membrane potential. After the first epoch, the hidden layer is completely evolved and no new neurons are added to the network. In this stage, the learning algorithm also updates the weights of the hidden neurons for multiple epochs when the criterion for 'margin maximization strategy for structure learning' is satisfied. In this strategy, the weights of the winner (neuron that fires first) neuron from the same class as the current input spike pattern are strengthened and the winner neuron from any other class are inhibited using the normalized membrane potential learning rule. The weight update using the normalized membrane potential learning rule ensures that the interclass margin (based on the response of hidden layer neurons) is maximized. This strategy is used until the average interclass margin for all the training spike patterns converges to a maximum value.

After the structure learning stage, the weights of the output layer neurons are initialized randomly. In the *output weights learning stage*, the weights of the output layer neurons are updated for multiple epochs using a similar margin maximization update mechanism until the average interclass margin (based on the response of output layer neurons) for all the training spike patterns converges to a maximum value.

TMM-SNN uses a heuristic criterion for 'neuron addition strategy' in the structure learning stage. This criterion relies on a threshold parameter for adding a new neuron to the network. To highlight the impact of this parameter on the performance of TMM-SNN, a study is conducted using the Ionosphere problem from the UCI machine learning repository [25]. Based on the results of this study, some guidelines have been presented for proper selection of this parameter. This study also illustrates the basic working of TMM-SNN.

Performance of TMM-SNN has been evaluated using ten different data sets, comprising both binary and multiclass problems, from the UCI machine learning repository. The performance evaluation has been conducted using ten random trials and the mean accuracies for ten random trials have been reported. A statistical study has also been conducted using the nonparameteric Friedman test followed by a pairwise comparison using the Fisher's least significant difference method for comparing the performance of TMM-SNN with that of SpikeProp [5], synaptic weight association training [22] and self-regulating evolving spiking neural classifiers [17]. Results of this statistical study clearly indicate that TMM-SNN performs better than the other learning algorithms with a 95% confidence interval using a compact network architecture.

The rest of the paper is organized as follows. Section II describes in detail the architecture and the learning algorithm for TMM-SNN. Section III analyzes the working of the TMM-SNN using the Ionosphere problem and also presents the results of performance comparison of TMM-SNN with other SNN learning algorithms. Section IV summarizes the conclusions from this study.

II. Two Stage Margin Maximizing Spiking Neural Network for Classification Problems

In this section, a Two stage Margin Maximization Spiking Neural Network (TMM-SNN) for classification problems is presented. First, the architecture of TMM-SNN is described followed by its learning algorithm.

A. Architecure of TMM-SNN

TMM-SNN employs a three layered SNN with an evolving hidden layer. The input layer is used to present the training spike patterns, one by one, $\{(\mathbf{x}_1, c_1), \cdots, (\mathbf{x}_n, c_n), \cdots, (\mathbf{x}_N, c_N)\}$ to the network. Each spike pattern is presented to the network for a duration T which is termed as the simulation interval. For all simulations in this paper, T has been set to 6 ms. The spike

pattern $\mathbf{x}_n = [x_n^1, \cdots, x_n^i, \cdots, x_n^m]$ is an m-dimensional spike pattern and $c_n \in \{1, \cdots, C\}$ is the corresponding class label. Here, C represents the total number of classes. Thus, the network has m input neurons and C output neurons. The input feature, $x_n^i = \{t_i^{(1)}, \cdots, t_i^{(g)}, \cdots, t_i^{(G_i^1)}\}$, is a spike train with G_i^1 spikes, where $t_i^{(g)}$ represents the g^{th} spike generated by the i^{th} input neuron. Note that, for simplifying the notation, the index of the current input spike pattern (n) has been suppressed from the time of spikes. Without loss of generality, it is assumed that the hidden layer has evolved to K neurons. Figure 1 shows the architecture of the TMM-SNN with K hidden neurons.

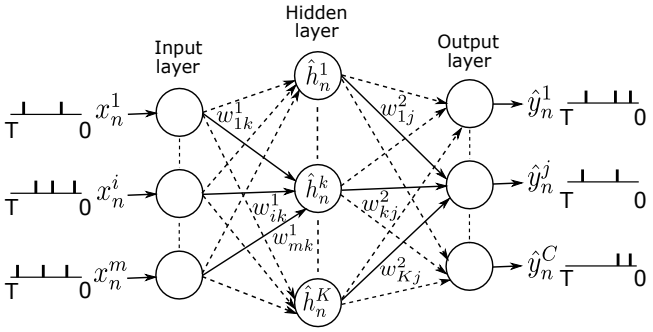


Fig. 1: Architecture of TMM-SNN

For a given input spike pattern ($\mathbf{x}_n$), the response of the k^{th} hidden neuron is a function of the membrane potential induced by the input neurons. The unweighted membrane potential ($\tilde{v}_i^1$) induced by the i^{th} input neuron at time t is given by

$$\tilde{v}_i^1(t) = \sum_g \epsilon\left(t - t_i^{(g)}\right) \tag{1}$$

where $\epsilon\left(t - t_i^{(g)}\right)$ is the unweighted membrane potential at time t that is induced by the presynaptic spike at $t_i^{(g)}$. In this work, $\epsilon(.)$ has been modeled using the spike response function [5], given by

$$\epsilon(s) = \frac{s}{\tau} \exp\left(1 - \frac{s}{\tau}\right) \tag{2}$$

where τ is the time constant for the neuron and is set to 3 ms for all the studies in this paper.

From Equation (1), the membrane potential (v_k^1) of the k^{th} hidden neuron at time (t) is given by

$$v_k^1(t) = \sum_i w_{ik}^{(1)} \tilde{v}_i^1(t); \qquad k \in \{1, \cdots, K\} \tag{3}$$

where $w_{ik}^{(1)}$ is the weight of the synapse between the i^{th} input neuron and the k^{th} hidden neuron. If θ_k^1 represents the threshold of the k^{th} hidden neuron, then the k^{th} hidden neuron generates a spike whenever its membrane potential crosses this threshold. The thresholds for all the hidden neurons are determined by the learning algorithm of TMM-SNN. After generating a spike, the membrane potential of the k^{th} hidden neuron is reset to zero. Thus, the response of the k^{th} hidden neuron for $\mathbf{x}_n$ is a spike train with G_k^2 spikes, denoted by $\hat{h}_n^k = \{\hat{t}_k^{(1)}, \cdots, \hat{t}_k^{(g)}, \cdots, \hat{t}_k^{(G_k^2)}\}$. Similarly, the spike pattern generated by the j^{th} output neuron depends on the membrane potential induced by the hidden layer neurons. The unweighted membrane potential ($\tilde{v}_k^2$) induced by the k^{th} hidden neuron at time t is given by

$$\tilde{v}_k^2(t) = \sum_g \epsilon(t - \hat{t}_k^{(g)}) \tag{4}$$

The total membrane potential (v_j^2) of the j^{th} output neuron at time t is given by

$$v_j^2(t) = \sum_k w_{kj}^{(2)} \tilde{v}_k^2(t); \qquad j \in \{1, \cdots, C\} \tag{5}$$

where $w_{kj}^{(2)}$ is the weight of the synapse between the k^{th} hidden neuron and the j^{th} output neuron. The j^{th} output neuron generates a spike when its membrane potential crosses its threshold (θ_j^2). The threshold for all the output neurons are set to unity. After generating a spike, the membrane potential of the neuron is reset to zero. Thus, the response of the j^{th} output neuron is a spike train with G_j^3 spikes and is denoted by $\hat{y}_n^j = \{\hat{t}_j^{(1)}, \cdots, \hat{t}_j^{(g)}, \cdots, \hat{t}_j^{(G_j^3)}\}$.

Based on the response of the output neurons, the predicted class label ($\hat{c}_n$) for a given input spike pattern ($\mathbf{x}_n$) is determined by the output neuron that fires first and is given by

$$\hat{c}_n = \underset{j}{\operatorname{argmin}}\, \hat{t}_j^{(1)} \tag{6}$$

Since, a correct prediction depends only on the time of the first spike generated by the output layer neurons, the learning algorithm utilizes only the time of the first spike generated by the hidden and output layer neurons in the learning process. Thus, the objective of the learning algorithm for TMM-SNN is to closely approximate the functional relationship between the input spike patterns and the corresponding class labels so that, for a given spike pattern, the correct class output neuron fires first. For convenience, the time of the first spike generated by the k^{th} hidden neuron and the j^{th} output neuron are represented as $\hat{t}_k^1$ and $\hat{t}_j^2$, respectively. Next, a description of the normalized membrane potential learning rule is provided.

B. Normalized Membrane Potential Learning Rule

The normalized membrane potential learning rule proposed in this paper utilizes the locally available information on a synapse and the membrane potential of a neuron to update the weights of a given synapse. It takes into account both the information present in the current input spike pattern as well as the knowledge acquired by the network from the past spike patterns to estimate the change in the weight of a synapse.

In the following description, the normalized membrane potential learning rule is explained for the synapses between the input layer neurons and the k^{th} hidden neuron (see Figure 1).

The normalized membrane potential learning rule uses the normalized membrane potential induced by the presynaptic neurons as a representation of the information present in a given input spike pattern. The normalized membrane potential (u_{ik}) of the i^{th} input neuron at time t is equal to the fraction of total unweighted membrane potential (ν_k) of the k^{th} hidden neuron that is induced by the i^{th} input neuron. The unweighted

membrane potential (ν_k) of the k^{th} hidden neuron at time t, given by

$$\nu_k(t) = \sum_i \tilde{v}_i^1(t) \tag{7}$$

Then the normalized membrane potential (u_{ik}) for the i^{th} input neuron at time t is computed as

$$u_{ik}(t) = \frac{\tilde{v}_i^1(t)}{\nu_k(t)}, \qquad i \in \{1, \cdots, m\} \tag{8}$$

It is also important to take into account the knowledge acquired by the network from the past input spike patterns while updating the weights of a given synapse. This helps in ensuring that the knowledge stored in the network is not lost as a result of updating the synaptic weights. For this purpose, the normalized membrane potential learning rule uses the existing weights of the synapses as they represent the consolidated effect of the past presynaptic inputs on the membrane potential of a hidden neuron.

Based on this, the normalized membrane potential learning rule updates the weight of the synapse between the i^{th} input neuron and the k^{th} hidden neuron using the difference between the normalized membrane potential ($u_{ik}(\hat{t}_k^1)$) of the i^{th} input neuron at $\hat{t}_k^1$ and the existing weight of a given synapse. Further, the weight of a synapse is only updated if its existing weight is lower than $u_{ik}(\hat{t}_k^1)$, otherwise its weight is not updated. The synapse having higher weights i.e. those synapse that have a significant contribution to the membrane potential for the past spike patterns are not updated. This ensures that the existing knowledge stored in the network is not lost after updating its weights for the current input spike pattern. Based on this, the appropriate change in the weight ($\Delta w_{ik}^{(1)}$) for the k^{th} hidden neuron is given by

$$\Delta w_{ik}^{(1)} = \begin{cases} 0 & w_{ik}^{(1)} \geq u_{ik}(\hat{t}_k^1) \\ \eta(u_{ik}(\hat{t}_k^1) - w_{ik}^{(1)}) & w_{ik}^{(1)} < u_{ik}(\hat{t}_k^1) \end{cases}, \; i \in \{1, \cdots, m\} \tag{9}$$

where η is the learning rate and in this paper it has been fixed at 0.01. The update rule in Equation (9) is referred to as the normalized membrane potential learning rule. Next, the normalized membrane potential learning rule is used to develop the two stage margin maximization learning algorithm for a spiking neural network.

C. Two Stage Margin Maximization Learning Algorithm

In this section, the normalized membrane potential learning rule described above is used to develop a two stage margin maximization learning algorithm for a spiking neural network referred to as a Two stage Margin Maximization Spiking Neural Network (TMM-SNN). The learning algorithm for TMM-SNN employs a two stage approach for learning. In the first stage (*structure learning stage*), the learning algorithm evolves the number of hidden neurons and estimates the hidden neuron parameters (synaptic weights and thresholds). In the second stage (*output weights learning stage*), the hidden neuron parameters are kept fixed and the synaptic weights of the output neurons are estimated. Next, the learning strategies used in the two stages of the learning algorithm are described in detail.

1) Structure Learning Stage: In this stage, the learning algorithm employs two different learning strategies, namely a 'neuron addition strategy' and a 'margin maximization strategy for structure learning'. In the first epoch, it uses 'neuron addition strategy' to completely evolve the number of hidden neurons and establish their associated classes. When a new neuron is added to the network, the class label of the input spike pattern used to add that neuron is stored as the associated class for that neuron. The 'margin maximization strategy for structure learning' is used to update the weights of the hidden neurons for multiple epochs such that the interclass margin based on the response of the hidden neurons is maximized. In the first epoch, a spike pattern is learnt using either a 'neuron addition strategy' or a 'margin maximization strategy for structure learning'. After the first epoch no new neurons are added to the network. In the subsequent epochs, the learning algorithm only updates the weights of existing neurons using the 'margin maximization strategy for structure learning'. Next, a description of these strategies is provided.

In the following description of the strategies, CC denotes the winner neuron (the neuron that fires first) from the same class as the current input spike pattern, and is given by

$$CC = \underset{k:\bar{c}_k = c_n}{\operatorname{argmin}} \hat{t}_k^1 \tag{10}$$

where $\bar{c}_k$ denotes the associated class for the k^{th} hidden neuron. Similarly, MC denotes the winner neuron from any other class, given by

$$MC = \underset{k:\bar{c}_k \neq c_n}{\operatorname{argmin}} \hat{t}_k^1 \tag{11}$$

Neuron addition strategy: In this strategy, the learning algorithm identifies the spike patterns for which the neuron CC fires late in the simulation interval. This indicates that the network has not acquired the information present in these spike patterns from the previously learnt spike patterns. Hence, a new neuron is added to the network to accurately approximate the information present in a given input spike pattern ($\mathbf{x}_n$). The class label (c_n) of the given input spike pattern (x_n) is stored as the associated class for the newly added neuron.

To identify the spike patterns for which a new neuron should be added to the network, a fixed time instant, $T_a \in [0, T]$ is used as a threshold for $\hat{t}_{CC}^1$. Based on this, the criterion for 'neuron addition strategy' is given by

$$\begin{aligned} &\textbf{If } \hat{t}_{CC}^1 > T_a \\ &\textbf{Then } \text{add a neuron to the network} \end{aligned} \tag{12}$$

where T_a is given as

$$T_a = \alpha_a T \tag{13}$$

Here, α_a is termed as the addition threshold and is always set in the interval $[0, 1]$. When α_a is set to one, T_a is equal to T, as a result, fewer neurons are added to the network. This leads to the development of an inaccurate model of the data. When α_a is set to zero, a neuron is added to the network for each input spike pattern which may result in overfitting. Therefore, the value for α_a needs to be chosen carefully and a suitable range for setting α_a is in the interval $[0.4, 0.55]$.

Fig. 2: Illustrative example for neuron addition strategy. A new neuron is added to the network whenever $\hat{t}^1_{CC}$ lies in the shaded region.

Figure 2 provides an illustrative example for 'neuron addition strategy'. It can be observed from the figure that, for a given spike pattern ($\mathbf{x}_n$), a new neuron is added to the network when $\hat{t}^1_{CC}$ lies in the interval $[T_a, T]$ (shaded region).

The weights and threshold of the newly added neuron are initialized such that the newly added neuron spikes precisely at $T/3$ for the current input spike pattern. For this purpose, the weights of the newly added neuron are initialized using the normalized membrane potential (Equation 8) of the input neurons at $T/3$ and is given by

$$w^{(1)}_{i(K+1)} = u_{ik}(T/3), \qquad i \in \{1, \cdots, m\} \tag{14}$$

After weight initialization, the threshold of the newly added neuron is initialized as

$$\theta^1_{(K+1)} = \sum_i w^{(1)}_{i(K+1)} \tilde{v}^1_i(T/3) \tag{15}$$

Margin maximization strategy for structure learning: In this strategy, the weights of the hidden neurons are updated such that the interclass margin (Υ^1_n) is maximized. For a given spike pattern ($\mathbf{x}_n$), the interclass margin (Υ^1_n) is computed as

$$\Upsilon^1_n = \hat{t}^1_{MC} - \hat{t}^1_{CC} \tag{16}$$

A smaller interclass margin for $\mathbf{x}_n$ indicates that it is closer to the decision boundary which may affect the performance of the learning algorithm on unseen spike patterns. To identify the spike patterns with lower Υ^1_n, the learning algorithm uses a time duration, T_m as a threshold for Υ^1_n. Thus, the criterion for this strategy is given as

If $\Upsilon^1_n < T_m$

Then update the weights of existing hidden neurons (17)

Since the weights and threshold of a newly added neuron are initialized such that it fires at $T/3$, T_m is set to a value in the interval $[T/3, T]$ and is given as

$$T_m = \alpha_m \left(T - \frac{T}{3} \right) \tag{18}$$

Here, α_m is termed as the margin threshold and it is always set in the interval $[0, 1]$. When T_m is set to a higher value, the learning algorithm tries to maintain a higher interclass margin which may result in the loss of knowledge learnt from the past spike patterns. Similarly, when T_m is set to a smaller value, the learning algorithm is not able to improve the interclass margin which may affect the performance of the learning algorithm on unseen spike patterns. It was observed that the performance of the learning algorithm is satisfactory when α_m is set in the interval $[0, 0.1]$. For all the simulation results reported in this paper, α_m is fixed at 0.07.

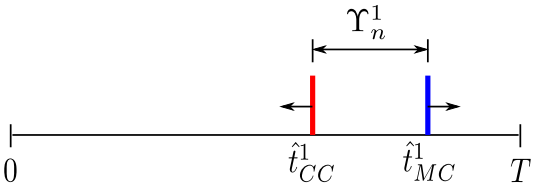


Fig. 3: Illustrative example for computation of interclass margin. The arrows depict the direction of shift in spike times of the neurons CC and MC when their weights are updated using the normalized membrane potential learning rule.

Algorithm 1 Pseudocode for the structure learning stage.

Evolving loop (Epoch 1)
for each training spike pattern **do**
  if $\hat{t}^1_{CC} > T_a$ **then**
    Add a neuron
  end if
end for
Iterative loop (Epoch 1 until convergence)
for each training spike pattern **do**
  if $(\hat{t}^1_{MC} - \hat{t}^1_{CC}) < T_m$ **then**
    Update weights of the neurons CC and MC
  end if
end for

Figure 3 provides an illustrative example for computation of interclass margin. It can be observed from the figure that for a given spike pattern ($\mathbf{x}_n$) the interclass margin (Equation (16)) depends only on the spike times of the neurons CC and MC. Hence, in this strategy the learning algorithm only updates the weights of the neurons CC and MC using the normalized membrane potential learning rule (Equation (9)). The weights of the neurons CC and MC are updated as

$$w^{(1)}_{i(CC)} = w^{(1)}_{i(CC)} + \Delta w^{(1)}_{i(CC)} \tag{19}$$

$$w^{(1)}_{i(MC)} = w^{(1)}_{i(MC)} - \Delta w^{(1)}_{i(MC)} \tag{20}$$

It may be noted that the weights of the neuron CC and MC are updated such that CC fires earlier than $\hat{t}^1_{CC}$ and MC fires later than $\hat{t}^1_{MC}$ respectively. This ensures that the interclass margin is increased when the weights of the neurons CC and MC are updated using the Equation (19) and (20), respectively.

The 'margin maximization strategy for structure learning' is used for multiple epochs to update the weights of the hidden neurons until the average interclass margin for all the training spike patterns converges to a maximum value.

The structure learning stage of the learning algorithm has been summarized in a pseudocode format in Algorithm 1. In the pseudocode, the evolving loop highlights the use of 'neuron addition strategy' in the first epoch to completely evolve the network architecture. The iterative loop depicts the use of 'margin maximization strategy for structure learning' for multiple epochs until the average interclass margin for all the training spike patterns converges to a maximum value.

2) Output weights learning stage: In the structure learning stage, the learning algorithm determines the number of hidden

neurons and their parameters (synaptic weights and threshold). At the end of this stage, the structure and the parameters of the hidden layer neurons are fixed. Based on the determined structure, the learning algorithm randomly initializes the weights of the output layer neurons. Then, the learning algorithm employs 'margin maximization strategy for output weights learning' for multiple epochs in order to maximize the interclass margin for the output layer neurons.

In this stage, the winner neurons (CC) from the same class as the current input spike pattern and the winner neuron (MC) from any other class are determined based on the spike times of the output layer neurons as given by

$$CC = \arg_{j} \ (j = c_n) \tag{21}$$

$$MC = \underset{j:j\neq c_n}{\operatorname{argmin}} \ \hat{t}_j^2 \tag{22}$$

Based on the spike times of the neurons CC and MC, the interclass margin (Υ_n^2) for the current input spike pattern ($\mathbf{x}_n$) is

$$\Upsilon_n^2 = \hat{t}_{MC}^2 - \hat{t}_{CC}^2 \tag{23}$$

Next, a description of 'margin maximization strategy for the output weights learning' stage is presented.

Margin maximization strategy for output weights learning: The purpose of this strategy is similar to that of 'margin maximization strategy for structure learning'. In this stage the learning algorithm updates the weights of the output layer neurons such that the interclass margin (Equation (23)) is maximized. Similar to the previous stage, the learning algorithm uses T_m as a threshold for the interclass margin to identify the spike patterns for which the weights of the output neurons should be updated. Thus, the criterion for this strategy is given by

If $\Upsilon_n^2 < T_m$

Then update the weights of the output neurons (24)

Since the interclass margin (Equation (23)) depends only on the spike times of the neurons CC and MC, the learning algorithm only updates the weights of these neurons.

After the first stage of the learning algorithm, the correct class hidden layer neuron fires first for most of the training spike patterns. Essentially, the second stage is used by the learning algorithm to map the response of hidden neurons back to the C-dimensional output space. For this reason, in this stage, the learning algorithm only updates the weights of the synapses between the hidden neurons associated with the class c_n and the output neurons. Furthermore, it uses the membrane potential induced by the corresponding hidden neuron for updating the synaptic weights to achieve faster convergence. Based on this, the appropriate change in weight ($\Delta w_{kj}^{(2)}$) for the output layer neurons is given by

$$\Delta w_{kj}^{(2)} = \begin{cases} 0 & \bar{c}_k \neq c_n \\ \eta(\tilde{v}_k^2(\hat{t}_j^2)) & \bar{c}_k = c_n \end{cases} \tag{25}$$

It may be observed from the above equation that the weights of the output layer neurons are also updated using locally available information as $\Delta w_{kj}^{(2)}$ relies only on the membrane potential induced by the hidden layer neurons.

Algorithm 2 Pseudocode for the output weights learning stage

Iterative loop (Epoch 1 until convergence)
for each training spike pattern **do**
  if $(\hat{t}_{MC}^2 - \hat{t}_{CC}^2) < T_m$ **then**
    Update weights of the neurons CC and MC
  end if
end for

Using Equation (25), the weights of the output neurons CC and MC are updated as

$$w_{k(CC)}^{(2)} = w_{k(CC)}^{(2)} + \Delta w_{k(CC)}^{(2)} \tag{26}$$

$$w_{k(MC)}^{(2)} = w_{k(MC)}^{(2)} - \Delta w_{k(MC)}^{(2)} \tag{27}$$

The weight update using Equations (26) and (27) ensures that CC fires earlier and MC fires late, thereby improving the interclass margin for the current input spike pattern.

The output weights learning stage of the learning algorithm is provided in a pseudocode format in Algorithm 2.

To summarise, the normalized membrane potential learning rule has been used to develop the two stage margin maximization learning algorithm for a spiking neural network. The learning algorithm for TMM-SNN has been summarized in a flowchart form in the Figure 4.

III. Performance Evaluation of the TMM-SNN

In this section, the performance of the Two stage Margin Maximization Spiking Neural Network is studied using ten benchmark classification problems from the UCI machine learning repository [25]. Performance of TMM-SNN is also compared with that of other existing batch learning algorithms for Spiking Neural Networks (SNNs), namely, SpikeProp [5], Synaptic Weight Association Training (SWAT) [22] and Self-Regulating Evolving Spiking Neural (SRESN) classifier [17] on all the problems.

The performance evaluation has been done based on three different metrics, namely, the overall training/testing accuracy, number of epochs required for convergence and the total number of estimated network parameters. Given that Q is the confusion matrix and q_{rs} represents the element in its r^{th} row and s^{th} column, the overall accuracy (η_a) is given by

$$\eta_a = \frac{100}{\sum_{r,s} q_{rs}} \sum_{r=1}^{C} q_{rr} \tag{28}$$

The total number of epochs is computed as the sum of the number of epochs required in the first and second stages of the TMM-SNN. The total number of estimated network parameters (η_p) is given by

$$\eta_p = md_hK + Kd_oC \tag{29}$$

where m is the number of input neurons, K is the number of hidden neurons and C is the total number of classes. d_h and d_o represent the number of delays used in the hidden layer and output layer respectively.

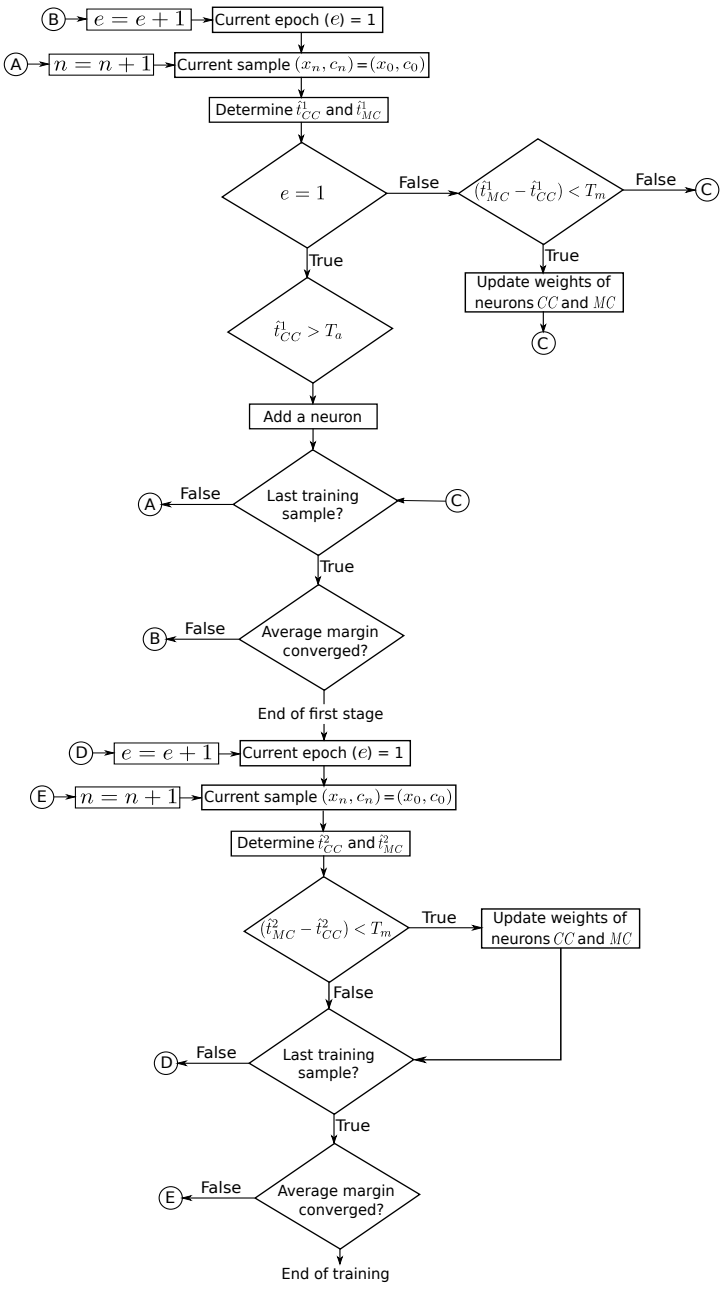
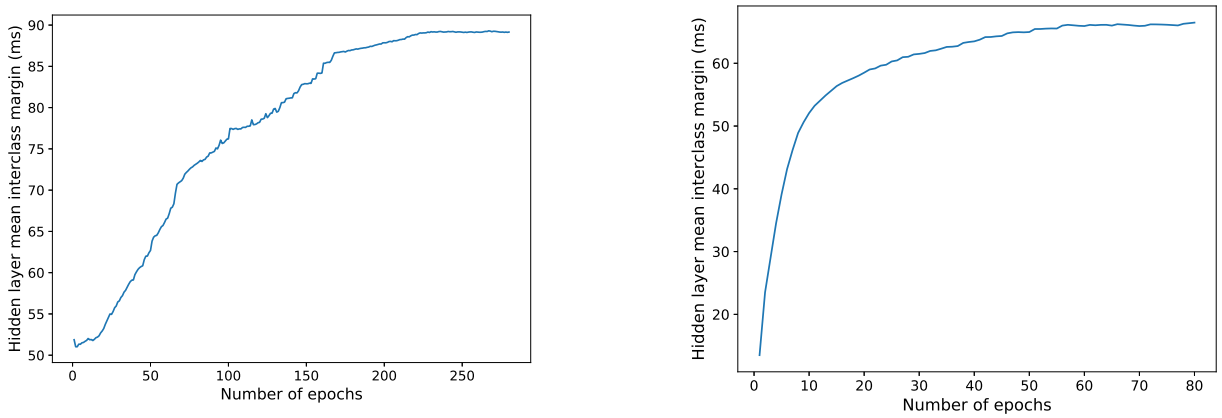


Fig. 4: Flowchart for the TMM-SNN learning algorithm

Performance results for all the algorithms have been generated using MATLAB 2014b in a Windows environment on a machine with 12 logical cores, 16 GB memory and a speed of 3.2 GHz. All the results reported in this section represent the average training/testing accuracy obtained over ten random trials. The real valued data from the data sets has been encoded into the spike patterns using the well known population coding [5], [26], [14] scheme. As in [27], the overlap constant for population coding is fixed at 0.7 and six receptive fields are used to encode a single feature for all problems. An encoding interval of 3 *ms* is used for population coding. Traditionally, population coding has been used with a fixed number of delays per connection by SpikeProp and other algorithms. This has a significant impact on the number of network parameters to be estimated. But, TMM-SNN does not employ any delays with population coding which implies that TMM-SNN uses a single synapse per connection in the network. The training procedure for the TMM-SNN is stopped in both the stages when the mean interclass margin for all the training samples remains constant within ± 1 for 50 epochs. During the structure



(a) (b)

Fig. 5: Effect of margin maximization strategy on the mean interclass margin for the response of (a) hidden layer neurons, (b) output layer neurons

learning stage, the mean interclass margin ($\bar{\Upsilon}^1$) is given by

$$\bar{\Upsilon}^1 = \frac{1}{N}\sum_n \Upsilon_n^1 \tag{30}$$

where N is the total number of samples in training. Similarly, during the output weights learning stage, the mean interclass margin ($\bar{\Upsilon}^2$) is given by

$$\bar{\Upsilon}^2 = \frac{1}{N}\sum_n \Upsilon_n^2 \tag{31}$$

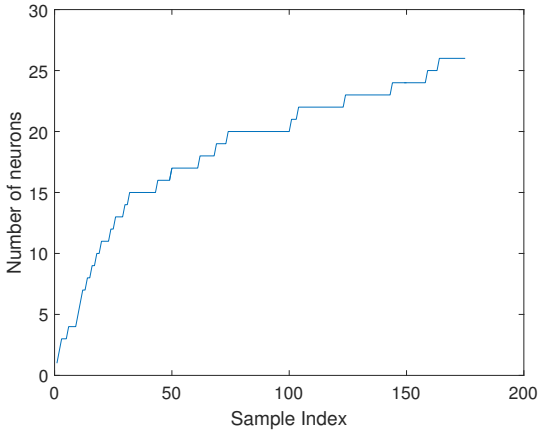


Fig. 6: Neuron growth history of TMM-SNN for the Ionosphere problem

A. Ionosphere Problem

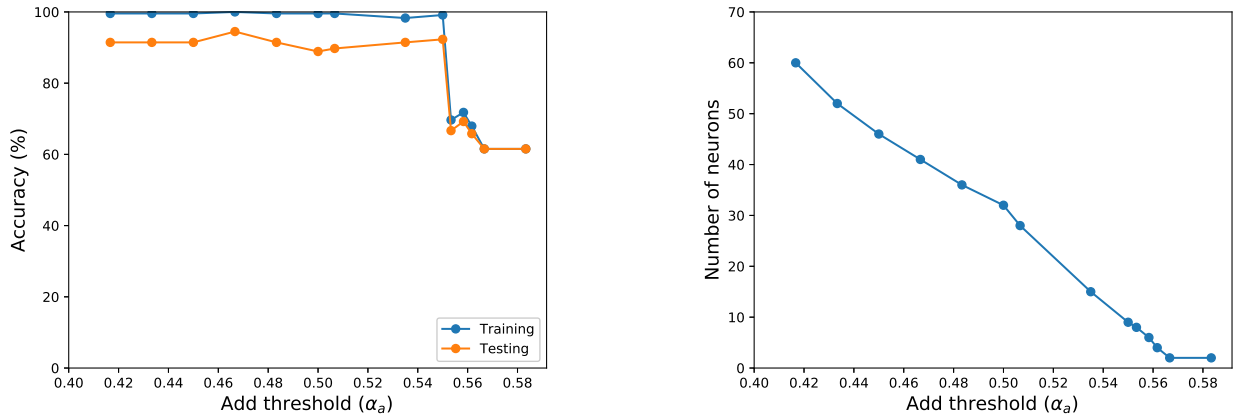
The Ionosphere problem consists of radar data and the goal here is to determine which of the samples provide information about the structure of the Ionosphere. The total number of samples is 351, out of which 175 samples are used for training and the rest are used for testing. Each sample consists of 34 features.

The Ionosphere problem is used to highlight the effect of the margin maximization strategy on the mean interclass margin of TMM-SNN during the two stages of the learning algorithm. Also, the Ionosphere problem is used to study the effect of the addition threshold (α_a) on the performance of the learning algorithm.

Figure 5a shows the effect of 'margin maximization strategy for structure learning' on the mean interclass margin ($\bar{\Upsilon}^1$) during the structure learning stage. It can be observed from the figure that $\bar{\Upsilon}^1$ improves continuously until it converges to a maximum value. Similarly, Figure 5b shows the effect of 'margin maximization strategy for output weights learning' on the mean interclass margin ($\bar{\Upsilon}^2$) during the output weights learning stage. It can be observed from the figure that the learning algorithm requires fewer epochs for convergence in the second stage of the learning algorithm. The faster convergence is achieved in the second stage as, after the first stage, the hidden neuron associated with the same class as the current input spike pattern fires first with high interclass margin. Next, the effect of the addition threshold (α_a) on the performance of TMM-SNN is studied.

When the addition threshold (α_a) is set to 0.5, TMM-SNN has a training and testing accuracy of 97.14% and 97.16%, respectively. Figure 6 shows the neuron growth history for the TMM-SNN. It can be observed from the figure that the learning algorithm adds higher number of neurons during the beginning of the training and fewer neurons are added closer to the end of the training.

To illustrate the effect of addition threshold (α_a) on the performance of TMM-SNN, the training and testing accuracies of TMM-SNN are evaluated for different values of α_a. To minimize the effect of the difference in number of epochs on the performance of TMM-SNN the network was trained for a fixed number of epochs (1000 epochs) in each stage of the learning algorithm. Since the parameters (weights and threshold) of a newly added neuron are initialized such that it fires precisely at $T/3$, the minimum value of α_a should be



(a) (b)

Fig. 7: Effect of addition threshold on (a) training and testing accuracy of TMM-SNN, (b) number of neurons added by TMM-SNN

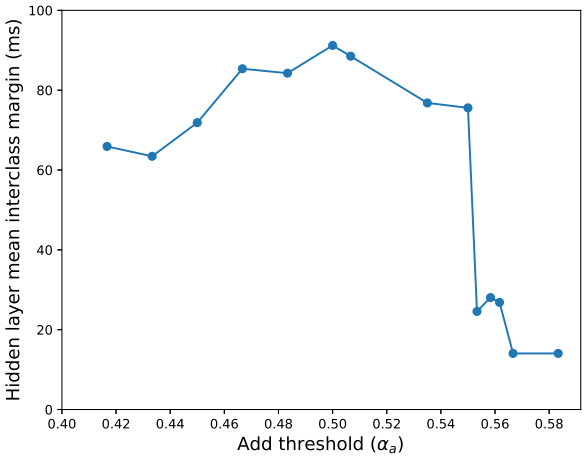


Fig. 8: Effect of addition threshold on the mean interclass margin for the hidden neurons

chosen such that T_a is greater than $T/3$. For this purpose, the performance of TMM-SNN are evaluated for different values of α_a in the interval $[0.4, 0.6]$. Figure 7a shows the effect of α_a on the training/testing accuracies of the TMM-SNN and Figure 7b shows the effect of α_a on the number of neurons added by the TMM-SNN during training. Figure 8 shows the effect of the addition threshold on the mean interclass margin of hidden layer neurons at the end of training. It can be seen from the Figure 7a that the performance of TMM-SNN does not vary significantly in the interval $[0.4, 0.55]$. Beyond 0.55, the performance of TMM-SNN deteriorates rapidly. For smaller values of α_a, large number of neurons are added to the network as a result the learning algorithm converges to a lower mean interclass margin (see Figure 8) for hidden neurons. For higher values of α_a, fewer neurons are added to the network and the learning algorithm converges to a higher mean interclass margin (see Figure 8) for hidden neurons. Thus, the margin

TABLE I: Description of the data sets used for evaluation

Data set	# Features	# Classes	# Samples	
			Training	Testing
Breast cancer	9	2	350	333
Echocardiogram	10	2	66	65
Mammogram	9	2	80	11
Liver	6	2	170	175
PIMA	9	2	384	384
Ionosphere	34	2	175	176
Hepatitis	19	2	78	77
Iris	4	3	75	75
Wine	13	3	60	118
Acoustic emission	5	4	62	137

based update mechanism of TMM-SNN helps the learning algorithm in achieving consistent performance for various network architectures. For very high values of α_a (greater than 0.55), the learning algorithm adds too few neurons to the network as a result performance of the learning algorithm deteriorates. Based on this, a suitable range for choosing α_a is the interval $[0.4, 0.55]$. It should be chosen closer to 0.4 for complex problems requiring large number of neurons and it should be chosen closer to 0.55 for simpler problems requiring fewer neurons.

B. Performance Comparison with Batch Learning Algorithms

In this section, the performance of TMM-SNN is evaluated using ten benchmark data sets from the UCI machine learning repository [25]. Results of the performance have been compared with other batch learning algorithms for SNNs, namely, SpikeProp [5], SWAT [22] and SRESN classifier [17]. Table I summarises the details of the data sets used for this evaluation. It provides information about the number of features, classes and the training/testing samples in a given data set.

Results for all the algorithms have been generated using the same training/testing data splits and the mean with standard deviation for overall training/testing accuracies over ten random trials is reported. As in the SpikeProp paper [5], the results for SpikeProp have been generated using a learning rate

of 0.0075, 16 delays per synapse, a coding interval of 4 ms and a time constant of 7 ms. For SpikeProp, the number of hidden neurons has been determined using the constructive-destructive procedure [28]. For SWAT, the parameters pertaining to the neuron model and those for frequency filtering have been set as in [22]. The parameters c_0 and the height of the plasticity window (A_p) in STDP are other important parameters for SWAT. As suggested in [22], the value for c_0 and A_p is fixed at 4000 and 0.1, respectively. The performance evaluation results of SRESN have been reproduced from [17]. Table II shows the value for the addition threshold (α_a) used for TMM-SNN on the benchmark data sets.

Table III shows the architecture and the results of performance evaluation for TMM-SNN, SpikeProp, SWAT and SRESN. The architecture is given in the format $(m-H-N)$. Here, m denotes the number of input neurons, H denotes the number of hidden neurons and N denotes the number of output neurons. Since SRESN and TMM-SNN evolve the number of hidden neurons, the number of hidden neurons in the architecture as well as the number of network parameters is shown as a range for the ten random trials. Further, the architecture for SRESN has been shown in the format $(m-H)$ as it employs a two layered SNN.

From the Table III, it may be noted that for simple binary classification problems like breast cancer and Echocardiogram, the training as well as testing accuracies of TMM-SNN are similar to that of other algorithms. For the breast cancer problem, TMM-SNN requires only 70 epochs for convergence whereas among other algorithms, SRESN requires a minimum of 306 epochs. For the Echocardiogram problem, TMM-SNN requires 177 epochs compared to 476 epochs required by SRESN.

For a low dimensional binary classification problem with a high interclass overlap like Mammogram and Liver, the training performance of TMM-SNN is 3%-5% better than that of SpikeProp which is the next best performing algorithm. A similar trend is also observed for the testing accuracy of TMM-SNN which also shows an improvement of 3%-5% over the testing accuracy of SpikeProp. Further, TMM-SNN converged in 176 epochs whereas SpikeProp was trained for 1000 epochs. PIMA is another low dimensional binary classification problem with a high interclass overlap. In case of PIMA, the training and testing accuracies of TMM-SNN are 1% and 2% better than that of SpikeProp. For this problem, TMM-SNN requires only 160 epochs whereas SpikeProp requires 3000 epochs. It may be noted, that for all these problems TMM-SNN requires lower than one-tenth the number of network parameters required by SpikeProp.

TABLE II: The value for the parameter addition threshold (α_a) used for evaluating TMM-SNN on the benchmark data sets

Data set	# Addition threshold (α_a)
Breast cancer	0.5
Echocardiogram	0.55
Mammogram	0.55
Liver	0.45
PIMA	0.45
Ionosphere	0.47
Hepatitis	0.55
Iris	0.42
Wine	0.5
Acoustic emission	0.45

Performance of TMM-SNN has also been evaluated for high dimensional problems of Ionosphere and Hepatitis. For Ionosphere, the training and testing accuracies of TMM-SNN are 7% and 4% better than that of SRESN, which is the next best performing algorithm. TMM-SNN and SRESN required 246 and 1018 epochs, respectively to achieve the reported accuracy. It may be observed that, among other algorithms SWAT had the best testing accuracy, but its performance results were inconsistent as its training accuracy is lower than its testing accuracy. For the Hepatitis problem, the training and testing accuracies of TMM-SNN are 4% and 3% better than that of SpikeProp, respectively. The reported accuracy is achieved by SpikeProp in 1000 epochs, whereas the TMM-SNN requires only 192 epochs.

For multiclass classification problems, performance of TMM-SNN is similar to the performance of SpikeProp. In comparison to SWAT and SRESN, the training accuracy of TMM-SNN is 4% better and the testing accuracy is 6% and 3% better, respectively. With regards to the number of epochs, TMM-SNN requires much lesser epochs to achieve convergence in comparison to the other algorithms used for comparison. Further, TMM-SNN achieves a much compact network architecture in comparison to the other algorithms used for comparison.

Statistical analysis of performance comparison: The results of performance evaluation of the learning algorithms SpikeProp, SWAT, SRESN and TMM-SNN have also been compared using the nonparameteric Friedman test followed by a pairwise comparison using the Fisher's least significant difference method. The non-parameteric Friedman test was conducted with the null hypothesis that the performance of all the different learning algorithms does not differ significantly from one another. If the p-value for the computed statistic is lesser than 0.05 than the null hypothesis is rejected with a 95% confidence interval. The test is conducted using the mean testing accuracies on the ten data sets described above. A p-value of 5e-4 is obtained which implies that the null hypothesis can be rejected with a 95% confidence interval. This shows that not all the algorithms perform equally. For further analysis, a pairwise test was conducted using the Fisher's Least Significant Difference (LSD) method. Using the LSD method, it is observed that TMM-SNN performs better than SWAT and SRESN with a level of significance equal to 0.05. For comparison with SpikeProp, the null hypothesis could be rejected with a level of significance equal to 0.06.

These observations clearly highlight that the margin based approach for tuning the network parameters helps the TMM-SNN to achieve a better or similar generalization performance in comparison to existing learning algorithms for spiking neural networks using a compact network structure. Further, the use of the normalized membrane potential learning rule to adjust the weights of the hidden layer neurons during the structure learning stage helps the TMM-SNN to converge faster in comparison to other existing learning algorithms for

TABLE III: Performance comparison of TMM-SNN with SpikeProp, SWAT and SRESN for benchmark problems

Data set	Learning Algorithm	Architecture	Training Accuracy (%)	Testing Accuracy (%)	No of Epochs	No of Network Parameters
Breast cancer	SpikeProp	55-15-2	97.3(0.6)	97.2(0.6)	1000	13680
	SWAT	54-702-2	96.5(0.5)	95.8(1.0)	500	1404
	SRESN	54-(8-12)	97.7(0.6)	97.2(0.7)	306	(432-648)
	TMM-SNN	54-(2-8)-2	97.4(0.3)	97.2(0.5)	70	(112-448)
Echocardiogram	SpikeProp	61-10-2	86.6(2.5)	84.5(3.0)	1000	10080
	SWAT	60-780-2	90.6(1.8)	81.8(2.8)	500	1560
	SRESN	60-(2-4)	79.1(5.7)	77.7(5.3)	476	(120-240)
	TMM-SNN	60-(2-3)-2	86.5(2.1)	85.4(1.7)	177	(124-186)
Mammogram	SpikeProp	55-10-2	82.8(4.7)	81.8(6.1)	1000	9120
	SWAT	54-702-2	82.6(2.1)	78.2(12.3)	500	1404
	SRESN	54-(4-8)	75.0(2.8)	76.6(4.5)	86	(216-432)
	TMM-SNN	54-(5-7)-2	87.2(4.4)	84.9(8.6)	176	(280-392)
Liver	SpikeProp	37-15-2	71.5(5.2)	65.1(4.7)	3000	9360
	SWAT	36-468-2	74.8(2.1)	60.9(3.2)	500	936
	SRESN	36-(6-9)	60.4(1.7)	59.7(1.7)	715	(216-324)
	TMM-SNN	36-(5-8)-2	74.2(3.5)	70.4(2.0)	442	(190-304)
PIMA	SpikeProp	55-20-2	78.6(2.5)	76.2(1.8)	3000	16640
	SWAT	54-702-2	77.0(2.1)	72.1(1.8)	500	1404
	SRESN	54-(9-14)	70.5(2.4)	69.9(2.1)	254	(486-756)
	TMM-SNN	54-(5-14)-2	79.7(2.3)	78.1(1.7)	160	(280-784)
Ionosphere	SpikeProp	205-25-2	89.0(7.9)	86.5(7.2)	3000	82800
	SWAT	204-2652-2	86.5(6.7)	90.0(2.3)	500	5304
	SRESN	204-(16-23)	91.9(1.8)	88.6(1.6)	1018	(3264-4692)
	TMM-SNN	204-(23-34)-2	98.7(0.4)	92.4(1.8)	246	(4738-7004)
Hepatitis	SpikeProp	115-15-2	87.8(5.0)	83.5(2.5)	1000	28080
	SWAT	114-1482-2	86.0(2.1)	83.1(2.2)	500	2964
	SRESN	114-(4-10)	79.8(0.9)	78.5(1.8)	234	(456-1140)
	TMM-SNN	114-(3-9)-2	91.2(2.5)	86.6(2.2)	192	(348-1044)
Iris	SpikeProp	25-10-3	97.2(1.9)	96.7(1.6)	1000	4480
	SWAT	24-312-3	96.7(1.4)	92.4(1.7)	500	936
	SRESN	24-(6-10)	96.9(1.0)	97.3(1.3)	102	(144-240)
	TMM-SNN	24-(4-7)-3	97.5(0.8)	97.2(1.0)	94	(108-189)
Wine	SpikeProp	79-10-2	99.2(1.2)	96.8(1.6)	1000	12960
	SWAT	78-1014-3	98.6(1.1)	92.3(2.4)	500	2028
	SRESN	78-(5-10)	96.9(1.6)	91.0(1.2)	128	(390-780)
	TMM-SNN	78-3-3	100(0)	97.5(0.8)	80	243
Acoustic emission	SpikeProp	31-10-4	98.5(1.7)	97.2(3.5)	1000	5600
	SWAT	30-390-4	93.1(2.3)	91.5(2.3)	500	1560
	SRESN	30-(4-8)	93.9(5.9)	94.2(3.2)	633	(120-240)
	TMM-SNN	30-(4-7)-4	97.6(1.3)	97.5(0.7)	12	(136-238)

TABLE IV: Description of the data sets used for comparison with SpikeTemp and eSNN

Data set	# Features	# Classes	# Samples	
			Training	Testing
Iris	4	3	90	60
Breast cancer	9	2	455	228
Liver	6	2	230	115
PIMA	8	2	512	256
Ionosphere	33	2	234	117
Yeast	8	10	990	494
Image Segmentation	18	7	210	2100
EEG eyestate	14	2	9990	4990

SNNs.

C. Performance Comparison with SpikeTemp and evolving Spiking Neural Network

In this section, the performance of TMM-SNN is compared with the performances of SpikeTemp [14] and evolving Spiking Neural Network (eSNN) [29], [10] on eight benchmark datasets from the UCI machine learning repository [25]. Table IV shows the details of the data sets used for comparison. It shows the details pertaining to the number of feature, classes and training/testing samples for a given data set.

The results for the learning algorithms SpikeTemp and eSNN have been reproduced from [14]. As in [14], the results for TMM were generated for a single fold of the data sets using the same number of training/testing samples. Table V shows the results of performance comparison between the performances of TMM-SNN, SpikeTemp and eSNN.

It can be observed from the table that for simple problems like Iris and breast cancer, the performance of TMM-SNN is similar to that of SpikeTemp and eSNN. However, to achieve this performance TMM-SNN uses a network with only 4 hidden neurons for both Iris and the breast cancer problem whereas, for the Iris problem, SpikeTemp and eSNN use a network with 87 and 84 neurons respectively and for the breast cancer problem, they use a network with 306 and 280 neurons respectively. It should also be noted that the number of neurons used by both SpikeTemp and eSNN are of the same order as the number of training samples whereas TMM-SNN uses a compact structure thereby increasing the generalization performance.

For difficult binary classification problems like Liver, PIMA

TABLE V: Performance comparison of TMM-SNN with SpikeTemp and eSNN

Data set	Learning Algorithm	Training Accuracy (%)	Testing Accuracy (%)	# Neurons
Iris	SpikeTemp	100	96.7	87
	eSNN	100	95	84
	TMM-SNN	97.8	98.3	4
Breast Cancer	SpikeTemp	99.1	98.3	306
	eSNN	99.6	98.7	280
	TMM-SNN	97.6	97.8	4
Liver	SpikeTemp	93	58.3	226
	eSNN	86.5	59.1	215
	TMM-SNN	77	71.3	10
PIMA	SpikeTemp	77.5	67.6	431
	eSNN	81	61.7	265
	TMM-SNN	78.9	77.7	10
Ionosphere	SpikeTemp	86.8	91.5	223
	eSNN	81.6	74.4	213
	TMM-SNN	99.1	94.0	7
Yeast	SpikeTemp	56.7	31.6	549
	eSNN	50.5	31.4	184
	TMM-SNN	59.3	62.4	11
Image Segmentation	SpikeTemp	89.1	82	174
	eSNN	71.9	70.9	191
	TMM-SNN	96.2	88.9	9
EEG eyestate	SpikeTemp	55.4	54.6	7
	eSNN	55.4	54.6	5
	TMM-SNN	55.1	55.2	6

and Ionosphere TMM-SNN performs 3%-13% better than SpikeTemp and 12%-20% better than eSNN. For the Liver and PIMA problems, TMM-SNN uses only 10 hidden neurons for each and for the Ionosphere problem it uses 7 neurons. Alternatively, SpikeTemp uses 226, 431 and 223 neurons for the Liver, PIMA and Ionosphere problems respectively. eSNN uses 215, 265 and 213 neurons for the Liver, PIMA and Ionosphere problems respectively. In this case also, the number of neurons required by these algorithms is of the order of the number of training samples whereas TMM-SNN uses a very compact network architecture with improved generalization.

For the multiclass problem of Yeast, TMM-SNN performs almost 31% better than SpikeTemp and eSNN. In this case, it uses only 11 neurons whereas SpikeTemp and eSNN employ 549 and 184 neurons each. Similar observations can also be deduced for the Image segmentation problem where 6% and 18% better than SpikeTemp and eSNN respectively. For the image segmentation problem, TMM-SNN uses 9 neurons whereas SpikeTemp and eSNN employ 174 and 191 neurons respectively.

For a problem with large number of samples like Eeg eyestate, the performance of the three algorithms is similar and they employ a similar architecture.

Statistical analysis of performance comparison: The performance of the learning algorithms SpikeTemp, eSNN and TMM-SNN are compared using the nonparameteric Friedman test followed by a pairwise comparison. The Friedman test is conducted using the null hypothesis that the performance of the different learning algorithms do not differ significantly. Based on the Friedman test, the null hypothesis is rejected with a p-value of 0.02. For further analysis, pairwwise comparisons are conducted on the performance evaluation results of TMM-SNN with SpikeTemp and eSNN using the Fishers Least Significant Difference method. In this case, the null hypothesis could be rejected with significance levels of 0.05 and 0.06 for the pairwise comparisons of TMM-SNN with eSNN and SpikeTemp, respectively.

This study clearly shows that TMM-SNN achieves better genaralization performance in comparison to SpikeTemp and eSNN using a compact network structure.

IV. Conclusions

In this paper, a Two stage Margin Maximization Spiking Neural Network (TMM-SNN) for pattern classification problems has been developed. TMM-SNN employs a three layered spiking neural network. It uses a two stage learning approach to evolve the number of hidden layer neurons and update the synaptic weights such that the interclass margin is maximized. For this purpose, a newly developed normalized membrane potential learning rule is used to update the synaptic weights based on the locally available information. Since, normalized membrane potential learning rule uses both the normalized membrane potential contributed by the presynaptic neurons and the past knowledge stored in the synapses, it is able to overcome the stability problem of local information based learning techniques. Further, the learning strategy to maximize the interclass margin helps in approximating the decision surface more accurately. The performance of TMM-SNN has been compared with other existing learning algorithms for SNNs using ten benchmark data sets from UCI machine learning repository. The results clearly indicate that TMM-SNN requires fewer epochs and smaller number of network parameters to obtain better performance. The statistical evaluation also indicates that TMM-SNN is better compared to other algorithm with a 95% confidence level.

References

[1] T. J. Sejnowski, "Time for a new neural code?" *Nature*, vol. 376, no. 6535, pp. 21–22, 1995.

[2] R. V. Rullen and S. J. Thorpe, "Rate coding versus temporal order coding: what the retinal ganglion cells tell the visual cortex," *Neural Computation*, vol. 13, no. 6, pp. 1255–1283, 2001.

[3] W. Maass, "Fast sigmoidal networks via spiking neurons," *Neural Computation*, vol. 9, no. 2, pp. 279–304, 1997.

[4] ——, "Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons," *Institute for Theoretical Computer Science, Technische Universitaet Graz, Austria, Technical Report*, 1999. [Online]. Available: http://www.igi.tugraz.at/psfiles/90.pdf

[5] S. M. Bohte, J. N. Kok, and H. L. Poutré, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1-4, pp. 17–37, 2002.

[6] S. B. Shrestha and Q. Song, "Adaptive learning rate of SpikeProp based on weight convergence analysis," *Neural Networks*, vol. 63, pp. 185–198, 2015.

[7] S. Ghosh-Dastidar and H. Adeli, "A new supervised learning algorithm for multiple spiking neural network with application in epilepsy and seizure detection," *Neural Networks*, vol. 22, no. 10, pp. 1419–1431, 2009.

[8] Y. Xu, X. Zeng, L. Han, and J. Yang, "A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks," *Neural Networks*, vol. 43, pp. 99–113, 2013.

[9] S. Ghosh-Dastidar and H. Adeli, "Improved spiking neural network for EEG classification and epilepsy and seizure detection," *Integrated Computer-Aided Engineering*, vol. 14, no. 3, pp. 187–212, 2007.

[10] K. Dhoble, N. Nuntalid, G. Indiveri, and N. Kasabov, "Online spatio-temporal pattern recognition with evolving spiking neural networks utilising address event representations, rank order, and temporal spike learning," in *International Joint Conference on Neural Networks*, 2012, pp. 1–7.

[11] S. G. Wysoski, L. Benuskova, and N. Kasabov, "Evolving spiking neural networks for audiovisual information processing," *Neural Networks*, vol. 23, no. 7, pp. 819–835, 2010.

[12] N. Kasabov, K. Dhoble, N. Nuntalid, and G. Indiveri, "Dynamic evolving spiking neural networks for on-line spatio-and spectro-temporal pattern recognition," *Neural Networks*, vol. 41, pp. 188–201, 2013.

[13] S. Dora, S. Ramasamy, and S. Sundaram, "A Basis Coupled Evolving Spiking Neural Network with Afferent Input Neurons," in *International Joint Conference on Neural Networks*, 2013, pp. 1–8.

[14] J. Wang, A. Belatreche, L. P. Maguire, and T. M. McGinnity, "Spiketemp: An enhance rank-order-based learning approach for spiking neural networks with adaptive structure," *IEEE Transaction on Neural Networks and Learning Systems*, 2015.

[15] S. Dora, S. Suresh, and N. Sundararajan, "A Sequential Learning Algorithm for a Minimal Spiking Neural Network (MSNN) Classifier," in *International Joint Conference on Neural Networks*, 2014, pp. 2415–2421.

[16] S. Dora, S. Sundaram, and N. Sundararajan, "A Two Stage Learning Algorithm for a Growing-Pruning Spiking Neural Network for Pattern Classification Problems," in *International Joint Conference on Neural Networks*, 2015.

[17] S. Dora, K. Subramanian, S. Suresh, and N. Sundararajan, "Development of a self-regulating evolving spiking neural network for classification problem," *Neurocomputing*, vol. 171, pp. 1216–1229, 2016.

[18] S. Soltic and N. Kasabov, "Knowledge extraction from evolving spiking neural networks with rank order population coding," *International journal of neural systems*, vol. 20, no. 6, pp. 437–45, 2010.

[19] S. Schliebs and N. Kasabov, "Evolving spiking neural networka survey," *Evolving Systems*, vol. 4, no. 2, pp. 87–98, 2013.

[20] J. Principe, N. R. Euliano, and W. C. Lefebvre, *Neural and Adaptive Systems: Fundamentals through Simulations*. Wiley, 1999.

[21] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single neuron, Populations and Plasticity*. Cambridge University Press, 2002.

[22] J. J. Wade, L. J. McDaid, J. A. Santos, and H. M. Sayers, "SWAT: A Spiking Neural Network Training Algorithm for Classification Problems," *IEEE Transactions on Neural Networks*, vol. 21, no. 11, pp. 1817–1830, 2010.

[23] E. L. Bienenstock, L. N. Cooper, and P. W. Munro, "Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex," *The Journal of neuroscience*, vol. 2, no. 1, pp. 32–48, 1982.

[24] F. Ponulak and A. Kasiński, "Supervised learning in spiking neural networks with resume: Sequence learning, classification and spike shifting," *Neural Computation*, vol. 22, no. 2, pp. 467–510, 2010.

[25] M. Lichman, "UCI Machine Learning Repository," 2013.

[26] J. Wang, A. Belatreche, L. Maguire, and T. M. McGinnity, "An online supervised learning method for spiking neural networks with adaptive structure," *Neurocomputing*, vol. 144, pp. 526–536, 2014.

[27] S. Dora, S. Suresh, and N. Sundararajan, "Online meta-neuron based learning algorithm for a spiking neural classifier," *Information Sciences*, vol. 414, pp. 19–32, 2017.

[28] S. Suresh, S. Omkar, V. Mani, and T. G. Prakash, "Lift coefficient prediction at high angle of attack using recurrent neural network," *Aerospace Science and Technology*, vol. 7, no. 8, pp. 595–602, 2003.

[29] S. G. Wysoski, L. Benushkova, and N. Kasabov, "Online learning with structural adaptation in a network of spiking neurons for visual pattern recognition," in *International Conference on Artificial Neural Networks*, 2006, pp. 61–70.